1. Basic DFAs

Let $\Sigma = \{\texttt{a}, \texttt{b}\}$. Draw a DFA for the following languages over Σ. The DFA must accept every string in the language and reject every string not in the language.

a. $\varnothing$

b. Σ^*

c. $\{\varepsilon\}$

d. $\overline{L}$, where $L = \{\texttt{baa}\}$ (Hint: First write out a DFA for L, then take the complement.)

2. DFAs, States, and Memory

One of the best starting points for designing a DFA is that the DFA's only "memory" is the state it's in at any given point. When designing DFAs, you can make every state correspond to some piece of information the automaton must "remember" about the string it's seen so far.

Consider this language over $\Sigma = \{\texttt{a}, \texttt{b}\}$, which we'll call L:

$$\{w \in \Sigma^* \mid w \text{ has an odd number of } a\text{'s and a number of } b\text{'s that is a multiple of 3}\}$$

a. When doing a DFA design problem, it's a good strategy to come up with strings to test a DFA on. Write down three strings from Σ^* that are in L and three strings from Σ^* that are not in L.

b. Another starting point is to work on a simpler version of the problem. Draw out a DFA for this simpler language:

$$\{w \in \Sigma^* \mid w \text{ has an odd number of } a\text{'s}\}$$

Then, think about what information each state represents.

c. What pieces of information would a DFA for L need to know/remember about the string that it's seen so far? Hint: think about the DFA from class for the language of strings involving a modular congruence. You should have six possible states.

d. Now, add one state for each piece of information the DFA would need to remember. Then, add in transitions based on how the information changes upon seeing one more character. Then, mark the appropriate states as accepting or not.

Here are some additional languages you can practice building DFAs for:

e. $\{w \in \Sigma^* \mid |w| > 1$ and the first and last character of w are the same $\}$

f. The set of strings with at least one `b`, but no more than one `a`.

3. Building NFAs

Design an NFA for these languages:

a. $L = \{w \in \Sigma^* \mid w \text{ ends in } \texttt{cab}\}$ over $\Sigma = \{\texttt{a}, \texttt{b}, \texttt{c}\}$.

b. $L = \{w \in \Sigma^* \mid$ there is a character in Σ that appears at most twice in $w\}$, over $\Sigma = \{\texttt{a}, \texttt{b}, \texttt{c}\}$. You'll want to use the guess-and-check model here, since any DFA for this language will require at least 64 states! (Hint: First try doing this with a smaller alphabet like $\{\texttt{a}, \texttt{b}\}$. What would you do if you knew what character was going to appear at most twice?)

4. Kleene Star and Language Concatenation Practice

Let's practice applying the formal definitions of language concatenation and the Kleene star:

$$L_1 L_2 = \{w \mid \exists x \in L_1.\ \exists y \in L_2.\ w = xy\}$$

$$L^0 = \{\varepsilon\} \qquad L^{n+1} = LL^n$$

$$L^* = \{w \mid \exists n \in \mathbb{N}.\ w \in L^n\}$$

a. Let $L = \{\texttt{abb}, \texttt{c}\}$. What is L^2? Give three examples of strings in L^*. Is $\varepsilon \in L^*$?

b. Let $L = \{\varepsilon\}$. What is L^*?

c. Let $L = \varnothing$. What is L^*?

d. Let $\Sigma = \{\texttt{a}, \texttt{b}\}$. Let L be the language $\{w \in \Sigma^* \mid |w| \not\equiv_3 2\}$. What is L^2? (Hint: Does this look like something from Problem Set 3?)

5. Proofs on Languages: The Finite Power Property

Since languages are sets of strings, proofs about languages will look similar to the proofs we wrote about sets. In particular, be mindful of the formal definitions listed earlier!

a. Let L be a language. Prove that for any natural number n, $L^n \subseteq L^*$. While L^n and L^* are languages, this is fundamentally the same as any subset problem we've previously tackled!

b. We say that a language L has the **finite power property** if the following statement is true:

$$\exists k \in \mathbb{N}.\ L^k = L^{k+1}$$

Come up with two languages that have the finite power property.

c. Let L be a language with the finite power property, and let k be a natural number where $L^k = L^{k+1}$. Prove by induction that for all natural numbers $n \geq k$, $L^k = L^n$.